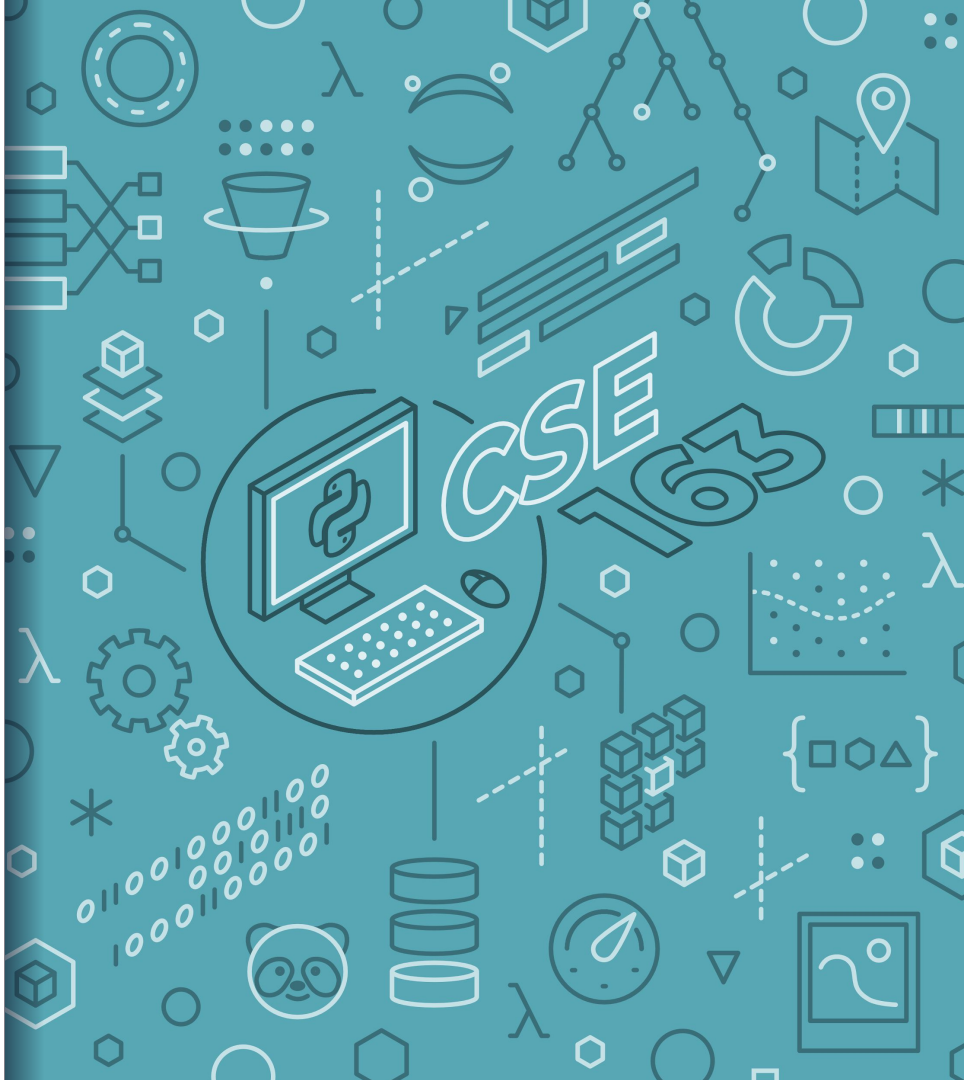
CSE 163

Dictionaries and Pandas

Hunter Schafer



Last Time

- More Lists
- Sets
- Dictionaries
- Tuples

This Time

- More Dictionaries
- CSVs
- Pandas

Recap

- Learned about 4 different data structures today!

1) **List**
   Has indices, this can be added/removed at any place
2) **Set**
   No duplicates, fast membership queries, no order
3) **Dictionary**
   Like a generic list that can have any value as keys
   Keys are unique, values are not
4) **Tuple**
   Immutable list, commonly used as returns, can “unpack”

Dictionary

- Python has a data structure called a **dictionary** that allows us to use arbitrary data as indices (keys)

```
d = {}
d['a'] = 1
d['b'] = 2
print(d)  # {'a': 1, 'b': 2}
```

- Dictionary Demo + Text Data Analysis Demo 2
  - Keys are unique, values are not
  - Basically a fancy list, most syntax is familiar
- Will start Wed by going more in-depth with dictionaries

Dictionary Methods

- Dictionary Methods

`dict() or {}`	Makes an empty dictionary
`d[key]`	Gets the value for k, KeyError if None
`d[key] = val`	Assigns val as the value for key
`d.pop(key)`	Removes key from this dictionary
`d.keys()`	Returns a collection of the keys
`d.values()`	Returns a collection of the values
`d.items()`	Returns a collection of (key, value) tuples

Comma Separated Values

CSV

- A file that encodes a spreadsheet. Separates cells w/ commas

Name	Salary
Erika	3
Erik	2
Josh	4

- If you were to save this as a CSV

```
Name,Salary
Erika,3
Erik,2
Josh,4
```

Processing CSVs

- Well structured so they are easy to parse!
- Not clear how to represent since we need info on rows and cols. Have to use multiple data structures!

```
Name,Salary
Erika,3
Erik,2
Josh,4
```

- We will use a **list of dictionaries** to store this information

```
[
    {'Name': 'Erika', 'Salary': '3'},
    {'Name': 'Erik',  'Salary': '2'},
    {'Name': 'Josh',  'Salary': '4'}
]
```

CSV Demo

- CSV Demo
  - Just showed the code for parse because it's not important to know how to write that code
  - Calculate the sum of TA Salaries
  - Work with earthquake data

id	year	month	day	latitude	longitude	name	magnitude
nc72666881	2016	7	27	37.6723333	-121.619	California	1.43
...	...	...	...	...	...	...	...

Brain Break

Processing CSVs

- A bit tedious
  - Code to read from the file is complicated (probably wrong too)
  - Have to pay particular attention to do the types
  - The code to compute statistics isn't too bad, but still have to write a bit
- If only there were a better way!

Pandas

- A **library** is a packaged version of someone else's code
- Pandas is a library that makes it super easy to work with CSVs

- To start, you can have pandas read the CSV for you

```
import pandas as pd
data = pd.read_csv('earthquakes.csv')
```

DataFrame

- One of the basic data types from pandas is a DataFrame
  - It's essentially a table with column and rows!

Columns

Index (row)

	id	year	month	day	latitude	longitude	name	magnitude
0	nc72666881	2016	7	27	37.672333	-121.619000	California	1.43
1	us20006i0y	2016	7	27	21.514600	94.572100	Burma	4.90
2	nc72666891	2016	7	27	37.576500	-118.859167	California	0.06

Series

- A Series is like a 1-dimensional DataFrame (no columns!)
  - Has an index
  - It's basically like a fancy dictionary/list hybrid
- For example

```
data['name']
```

0	California
1	Burma
2	California

```
data['name'][1]  # 'Burma'
```

Series Methods

- Series has a many methods that help you process your data
  - `mean`
  - `min`
  - `max`
  - `idxmin`
  - `idxmax`
  - `count`
  - `unique`
  - And many many more!
- Element-wise operators
  - Arithmetic: `+`, `-`, `*`, `/`, etc.
  - Comparison: `<`, `>`, `==`

Filtering

- Can use a bool Series to select which rows from the dataset

```
mask = data['magnitude'] > 5
data[mask]
# Same as: data[data['magnitude'] > 5]
```

	id	year	month	day	latitude	longitude	name	magnitude
30	us20006i18	2016	7	27	-24.286000	-67.864700	Chile	5.60
114	us20006i35	2016	7	27	36.492200	140.756800	Japan	5.30
421	us1000683b	2016	7	28	-16.824200	-172.515800	Tonga	5.10

- Can use multiple filters with: & (and), | (or), ~ (not)

```
data[(data['magnitude'] > 5) & ~(data['day'] == 27)]
```

Location Location Location

- When working with pandas, you have to think about
  1) How do I get to the data I want?
  2) Want computation do I want to do on that data?
- Very experimental, highly recommend using a Jupyter Notebook to try different things to see how to get the data

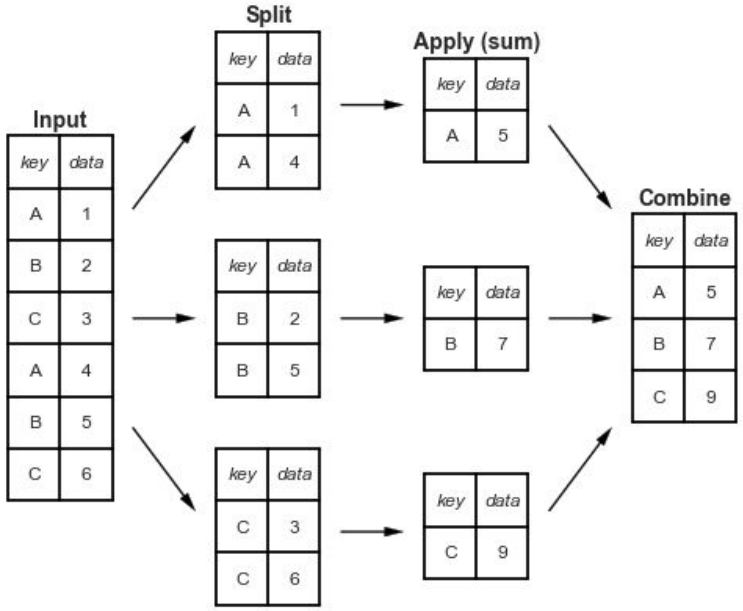
- So far, we have shown you how to access columns and filter
- These are special syntax for a more general method of accessing data

```
data.loc[<row indexer>, <column indexer>]
```

- Location Demo

Group By

- groupby lets you group and aggregate your data



- We will cover this more on Friday!